

Dynamic versus Static Polymorphism with C++


Content

	PLOYMORPHISM
Content	1
1 Content.....	3
2 Polymorphism.....	4
2.1 Introduction.....	4
2.2 Dynamic versus Static	5
3 Example Resource Protection	6
3.1 Architecture and Design.....	6
3.2 Implementation Approaches Overview.....	7
3.3 Interface and Association.....	8
3.3.1 Architecture and Design	8
3.3.2 Implementation Topics: Resource_Locker.....	9
3.3.3 Implementation Topics: Counter.....	10
3.3.4 Implementation Topics: Application	11
3.4 Template Parameter	12
3.4.1 Architecture and Design	12
3.4.2 Implementation Topics: Resource_Locker.....	13
3.4.3 Implementation Topics: Counter.....	14
3.4.4 Implementation Topics: Application	15
3.5 CRTP (Curiously Recurring Template Pattern)	16
3.5.1 Introduction	16
3.5.2 Implementation Topics: Resource Locker	17
3.5.3 Implementation Topics: Counter.....	18
3.5.4 Implementation Topics: Application	19
3.6 Approaches Comparison	20

3.6.1	Map File: Library	20
3.6.2	Map File: Application	21
3.6.3	Data.....	23
3.6.4	Trace	25
3.6.5	Summary	26
4	Summary and Outlook.....	27
5	MicroConsult Training and Coaching on Polymorphism	28

1 Content

Content
▪ Polymorphism - introduction
▪ Exemplary resource protection
▪ Implementation approach: interface and association
▪ Implementation approach: template parameter
▪ Implementation approach: CRTP (curiously recurring template pattern)
▪ Comparison of approaches including efficiency
▪ Summary and outlook

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S 2  MICROCONSULT

2 Polymorphism

2.1 Introduction

Polymorphism – Introduction

cA

```
- mData:uint32_t
+ cA()
+ ~cA()
+ function(Data:uint32_t)
```

cB

```
- mData:uint32_t
+ cB()
+ ~cB()
+ function(Data:uint32_t)
```


Polymorphism in the context of object oriented programming:

- Function with the same semantic,
- provided in more than one class
- with different function implementations.
- The different functions can be called **more or less** in the same way.

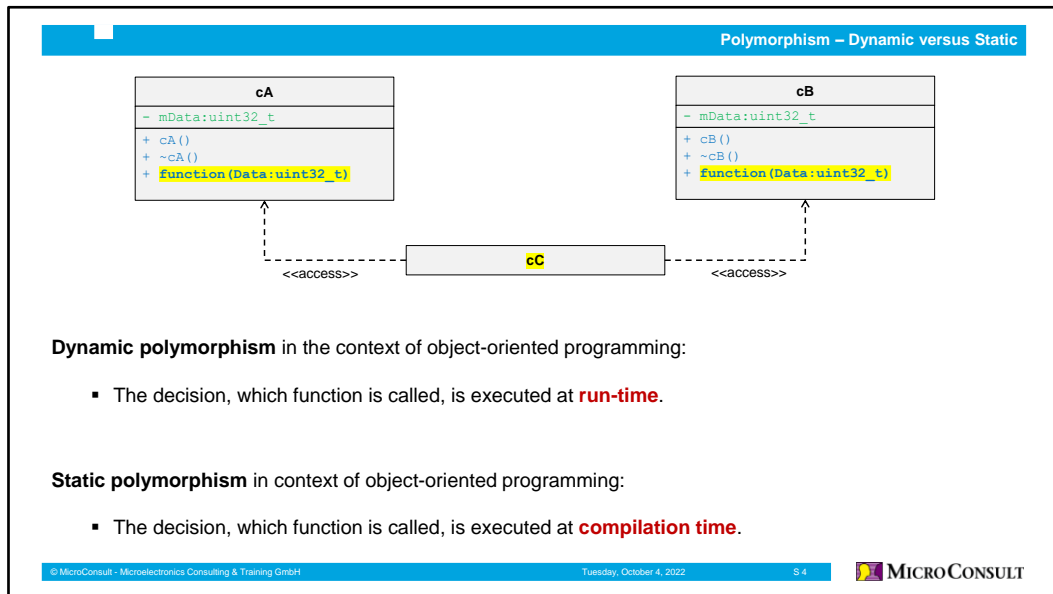
© MicroConsult - Microelectronics Consulting & Training GmbH

Tuesday, October 4, 2022

S 3

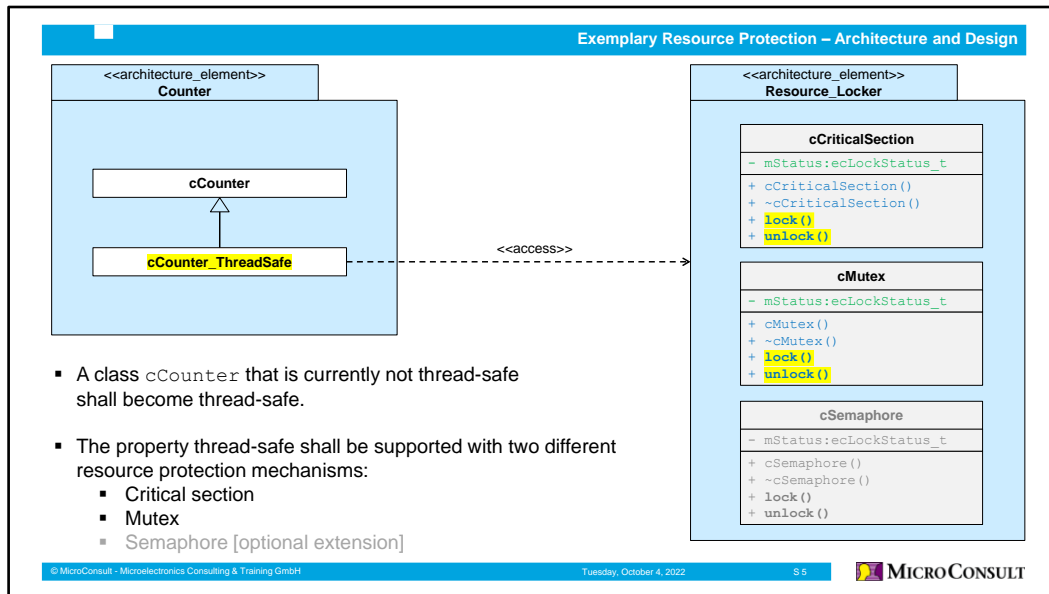
 MICROCONSULT

2.2 Dynamic versus Static

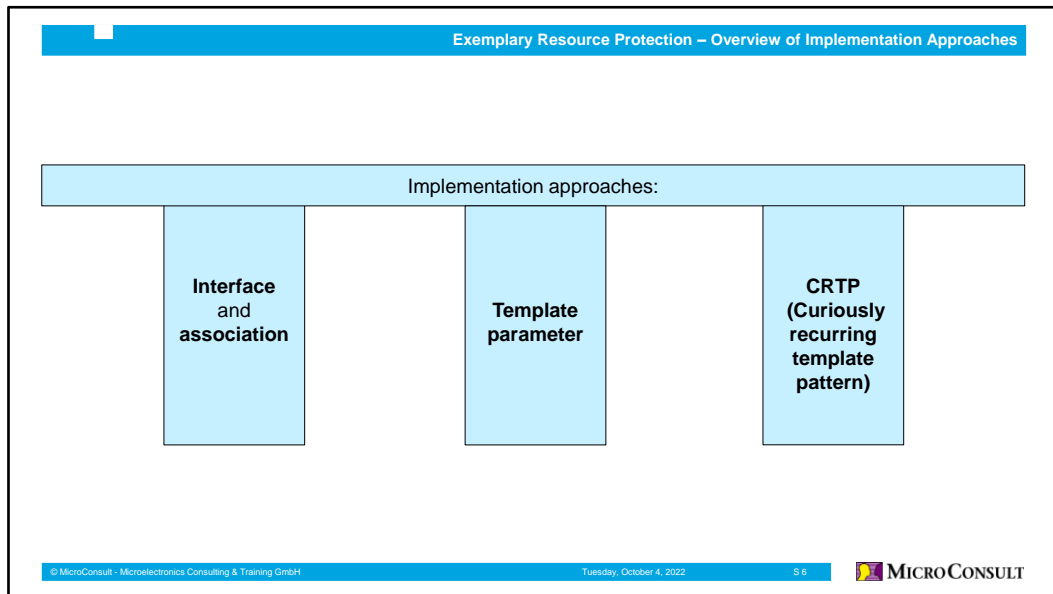


3 Example Resource Protection

3.1 Architecture and Design



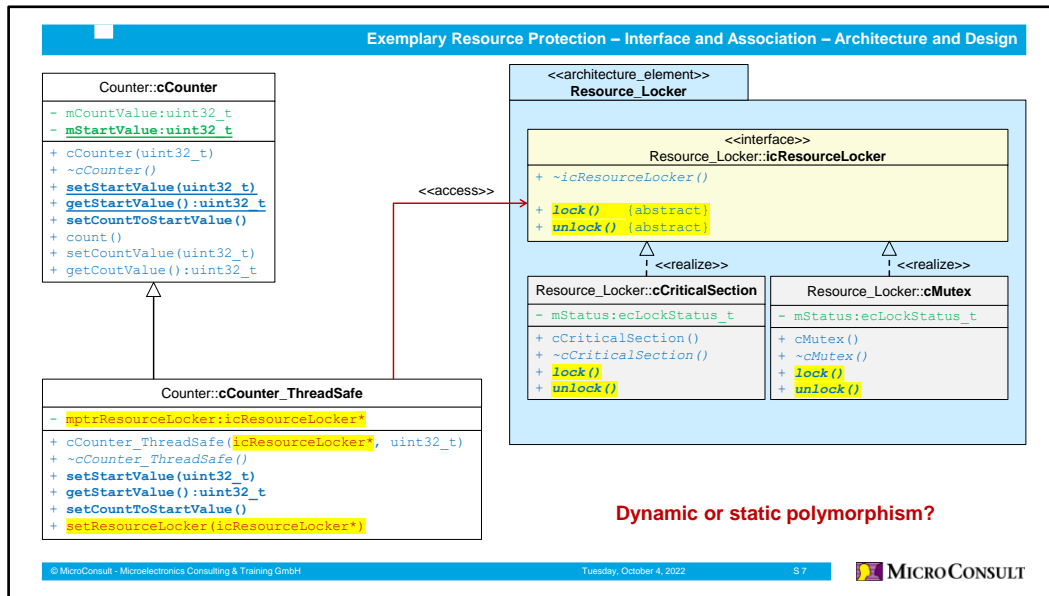
3.2 Implementation Approaches Overview



Example Resource Protection

3.3 Interface and Association

3.3.1 Architecture and Design



3.3.2 Implementation Topics: Resource_Locker

Exemplary Resource Protection – Interface and Association – Implementation Topics: Resource_Locker

```

class icResourceLocker
{
public:
    virtual ~icResourceLocker() =default;
    virtual void lock(void) =0;
    virtual void unlock(void) =0;
};
    
```

icResourceLocker.hpp

Interface class containing pure virtual (abstract) interface functions

```

class cMutex : public icResourceLocker
{
public:
    cMutex(void) =default;
    ~cMutex() override =default;

    void lock(void) override;
    void unlock(void) override;

private:
    ecLockStatus_t mStatus = ecLockStatus_t::Unlocked;
};
    
```

cMutex.hpp

Interface realization in the class cMutex, cCriticalSection looks similar.

```

void cMutex::lock(void)
{
    // ... operating system call
    mStatus = ecLockStatus_t::Locked;
    showValue("\nMutex status = locked");
}

void cMutex::unlock(void)
{
    // ... operating system call
    mStatus = ecLockStatus_t::Unlocked;
    showValue("\nMutex status = unlocked");
}
    
```

cMutex.cpp

Interface function implementations in the class cMutex. mStatus and showValue() are only for simulation.

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 58 MICROCONSULT

Example Resource Protection

3.3.3 Implementation Topics: Counter

Exemplary Resource Protection – Interface and Association – Implementation Topics: Counter

```

class cCounter_ThreadSafe : public cCounter
{
public:
    cCounter_ThreadSafe(icResourceLocker* const ptrResourceLocker = nullptr,
        uint32_t const CountValue = 0);
    ~cCounter_ThreadSafe() override =default;

    void setStartValue(uint32_t const StartValue);
    uint32_t getStartValue(void);
    void setCountToStartValue(void);

    void setResourceLocker(icResourceLocker* const ptrResourceLocker);

private:
    icResourceLocker* mptrResourceLocker;
};

```

Interface pointer that can be assigned to an object of type cCriticalSection or cMutex.

Redefinition of critical base class functions to make them thread-safe

Resource locker determination by setting the interface pointer

```

void cCounter_ThreadSafe::setCountToStartValue(void)
{
    if (mptrResourceLocker != nullptr)
    {
        mptrResourceLocker->lock();
    }

    cCounter::setCountToStartValue();

    if (mptrResourceLocker != nullptr)
    {
        mptrResourceLocker->unlock();
    }
}

```

Interface function call depending on the object type behind the pointer (dynamic / late binding)

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S 9 MICROCONSULT

3.3.4 Implementation Topics: Application

Exemplary Resource Protection – Interface and Association – Implementation Topics: Application

```
#include "Main_Application.hpp"
#include "../Counter/cCounter_ThreadSafe.hpp"
#include "../Resource_Locker/cCriticalSection.hpp"
#include "../Resource_Locker/cMutex.hpp"

int main(void)
{
    Resource_Locker::cCriticalSection locCriticalSection{ };
    Resource_Locker::cMutex locMutex{ };

    Counter::cCounter_ThreadSafe locCounter_A{ locCriticalSection };
    Counter::cCounter_ThreadSafe locCounter_B{ locMutex };

    locCounter_A.setStartValue(20);
    locCounter_B.setStartValue(30);

    locCounter_A.setResourceLocker(&locMutex);
    locCounter_B.setResourceLocker(&locCriticalSection);

    locCounter_A.setCountToStartValue();
    locCounter_B.setCountToStartValue();

    locCounter_A.count();
    locCounter_B.count();
}
```

Main_Application.cpp

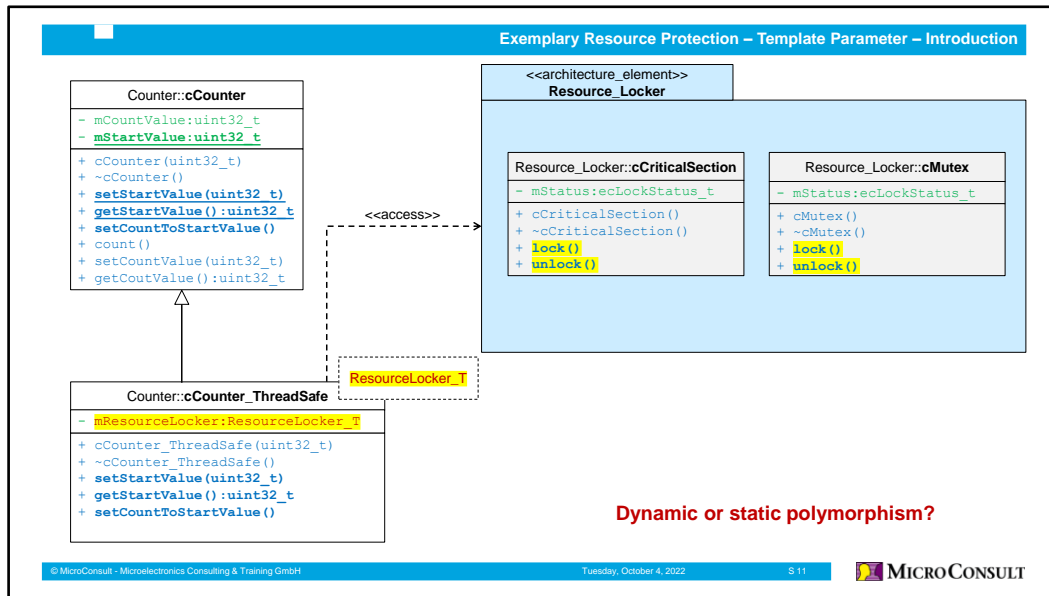
Initial resource locker assignment

Exchanging the resource lockers at run-time:
→ dynamic polymorphism

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S 10 MICROCONSULT

3.4 Template Parameter

3.4.1 Architecture and Design



3.4.2 Implementation Topics: Resource_Locker

Exemplary Resource Protection – Template Parameter – Implementation Topics: Resource_Locker

Interface function implementations in the class `cMutex`. `mStatus` and `showValue()` are only for simulation.

```
class cMutex
{
public:
    cMutex(void) =default;
    ~cMutex() =default;

    void lock(void);
    void unlock(void);

private:
    ecLockStatus_t mStatus = ecLockStatus_t::Unlocked;
};
```

cMutex.hpp

Function definition in the class `cMutex`, `cCriticalSection` looks similar.

```
void cMutex::lock(void)
{
    // ... operating system call
    mStatus = ecLockStatus_t::Locked;
    showValue("\nMutex status = locked");
}

void cMutex::unlock(void)
{
    // ... operating system call
    mStatus = ecLockStatus_t::Unlocked;
    showValue("\nMutex status = unlocked");
}
```

cMutex.cpp

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S 12 MICROCONSULT

3.4.3 Implementation Topics: Counter

Exemplary Resource Protection – Template Parameter – Implementation Topics: Counter

```

template<typename ResourceLocker_T>
class tcCounter_ThreadSafe : public cCounter
{
public:
    tcCounter_ThreadSafe(uint32_t const CountValue = 0);
    ~tcCounter_ThreadSafe() =default;

    void setStartValue(uint32_t const StartValue);
    uint32_t getStartValue(void);
    void setCountToStartValue(void);

private:
    ResourceLocker_T mResourceLocker;
};
    
```

cCounter_ThreadSafe.hpp

Redefinition of static base class functions to make them thread-safe

Object of template parameter ResourceLocker_T type determines the kind of resource locker.

```

template<typename ResourceLocker_T>
void tcCounter_ThreadSafe<ResourceLocker_T>::setCountToStartValue(void)
{
    mResourceLocker.lock();

    cCounter::setCountToStartValue();

    mResourceLocker.unlock();
}
    
```

cCounter_ThreadSafe.hpp

Function calls depend on the type of object mResourceLocker (static / early binding).

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S.13 MICROCONSULT

3.4.4 Implementation Topics: Application

Exemplary Resource Protection – Template Parameter – Implementation Topics: Application

```
#include "Main_Application.hpp"

#include "../Counter/cCounter_ThreadSafe.hpp"
#include "../Resource_Locker/cCriticalSection.hpp"
#include "../Resource_Locker/cMutex.hpp"

int main(void)
{
    Counter::tcCounter_ThreadSafe<Resource_Locker::cCriticalSection> locCounter_CriticalSection{ };
    Counter::tcCounter_ThreadSafe<Resource_Locker::cMutex> locCounter_Mutex{ };

    locCounter_CriticalSection.setStartValue(20);
    locCounter_Mutex.setStartValue(30);

    locCounter_CriticalSection.setCountToStartValue();
    locCounter_Mutex.setCountToStartValue();

    locCounter_CriticalSection.count();
    locCounter_Mutex.count();
}
```

Main_Application.cpp

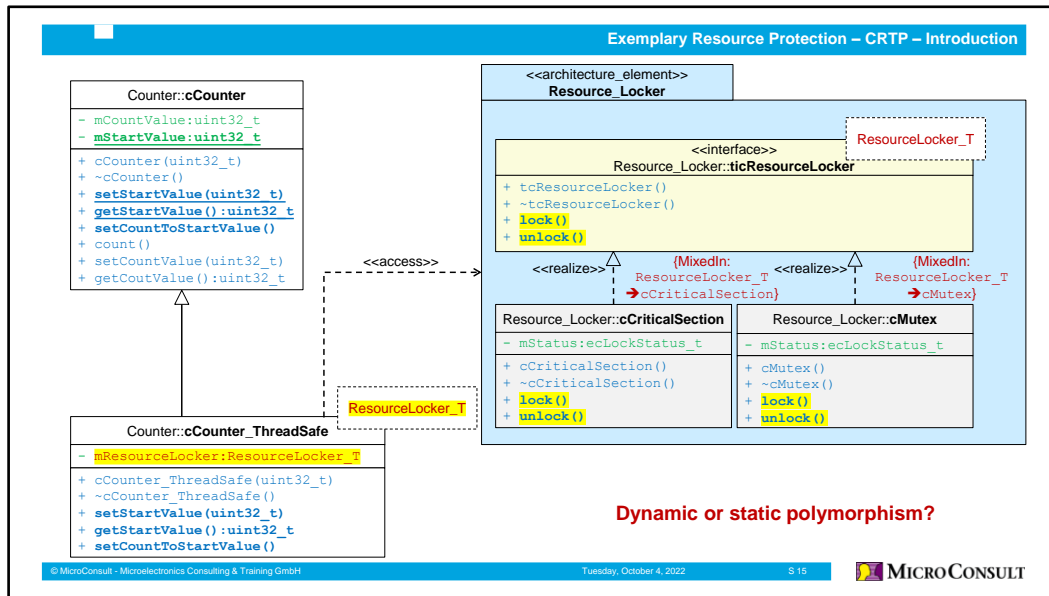
Resource lockers are determined at coding / compilation time and cannot be exchanged at runtime:
→ static polymorphism

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S 14 MICROCONSULT

Example Resource Protection

3.5 CRTP (Curiously Recurring Template Pattern)

3.5.1 Introduction



3.5.2 Implementation Topics: Resource Locker

Exemplary Resource Protection – CRTP – Implementation Topics: Resource_Locker

```

template<typename ResourceLocker_T>
class ticResourceLocker
{
public:
    ticResourceLocker(void) =default;
    ~ticResourceLocker() =default;

    void lock(void);
    void unlock(void);
};

template<typename ResourceLocker_T>
void ticResourceLocker<ResourceLocker_T>::lock(void)
{
    static_cast<ResourceLocker_T*>(this)->lock();
}

template<typename ResourceLocker_T>
void ticResourceLocker<ResourceLocker_T>::unlock(void)
{
    static_cast<ResourceLocker_T*>(this)->unlock();
}
    
```

ticResourceLocker.hpp

```

class cMutex : public ticResourceLocker<cMutex>
{
public:
    cMutex(void) =default;
    ~cMutex() =default;

    void lock(void);
    void unlock(void);

private:
    ecLockStatus_t mStatus = ecLockStatus_t::Unlocked;
};
    
```

cMutex.hpp

Template base class parameter is set to the derivate class
→ MixedIn

```

void cMutex::lock(void)
{
    // ... operating system call
    mStatus = ecLockStatus_t::Locked;
    showValue("\nMutex status = locked");
}

void cMutex::unlock(void)
{
    // ... operating system call
    mStatus = ecLockStatus_t::Unlocked;
    showValue("\nMutex status = unlocked");
}
    
```

cMutex.cpp

Interface class already contains interface function implementations as wrappers to the more specialized functions of the derived classes cCriticalSection and cMutex.

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S 16 MICROCONSULT

Example Resource Protection

3.5.3 Implementation Topics: Counter

Exemplary Resource Protection – CRTP – Implementation Topics: Counter

```

template<typename ResourceLocker_T>
class tcCounter_ThreadSafe : public cCounter
{
public:
    tcCounter_ThreadSafe(uint32_t const CountValue = 0);
    ~tcCounter_ThreadSafe() =default;

    void setStartValue(uint32_t const StartValue);
    uint32_t getStartValue(void);
    void setCountToStartValue(void);

private:
    ResourceLocker_T mResourceLocker;
};
    
```

cCounter_ThreadSafe.hpp

Redefinition of static base class function to make them thread-safe

Object of template parameter ResourceLocker_T type determines the kind of resource locker.

```

template<typename ResourceLocker_T>
void tcCounter_ThreadSafe<ResourceLocker_T>::setCountToStartValue(void)
{
    mResourceLocker.lock();

    cCounter::setCountToStartValue();

    mResourceLocker.unlock();
}
    
```

cCounter_ThreadSafe.hpp

Function calls depend on the type of object mResourceLocker (static / early binding).

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S. 17 MICROCONSULT

3.5.4 Implementation Topics: Application

Exemplary Resource Protection – CRTP – Implementation Topics: Application

```
#include "Main_Application.hpp"

#include "../Counter/cCounter_ThreadSafe.hpp"
#include "../Resource_Locker/cCriticalSection.hpp"
#include "../Resource_Locker/cMutex.hpp"

int main(void)
{
    using namespace Counter;
    using namespace Resource_Locker;

    tcCounter_ThreadSafe<tcResourceLocker<cCriticalSection>> locCounter_CriticalSection{ };
    tcCounter_ThreadSafe<tcResourceLocker<cMutex>> locCounter_Mutex{ };

    locCounter_CriticalSection.setStartValue(20);
    locCounter_Mutex.setStartValue(30);

    locCounter_CriticalSection.setCountToStartValue();
    locCounter_Mutex.setCountToStartValue();

    locCounter_CriticalSection.count();
    locCounter_Mutex.count();
}
```

Main_Application.cpp

Resource lockers are determined at coding / compilation time and cannot be exchanged at runtime:
→ static polymorphism

© MicroConsult - Microelectronics Consulting & Training GmbH

Tuesday, October 4, 2022

S 18

MICROCONSULT

3.6 Approaches Comparison

3.6.1 Map File: Library

Exemplary Resource Protection – Comparison of Approaches – Map File: Library							
	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	Library Name	
Interface and association	756	54	0	0	96	1300	c_w.l
	4	0	0	0	0	380	libcpp_w.l
	44	32	0	0	0	88	libcppabi_w.l
	812	86	0	0	96	1768	Library Totals
	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	Library Name	
Template parameter	512	16	0	0	96	684	c_w.l
	518	16	0	0	96	684	Library Totals
	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	Library Name	
CRTP	512	16	0	0	96	684	c_w.l
	518	16	0	0	96	684	Library Totals

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 8:19 MICROCONSULT

The efficiency analysis is based on:

- STM32F407 Microcontroller with SYSCLK = 168MHz
- MCBSTM32F400 Evaluation board
- MDK-ARM v5.35.0.2 Tool chain
- ARM Clang compiler v6.16
- Compiler optimization default
- C++14

Code : Program code
 Inc. Data : Include data
 RO Data : Constants
 RW Data : Initialized global data/objects
 ZI Data : Zero-initialized global data/objects
 Debug : Debug information

3.6.2 Map File: Application

Exemplary Resource Protection – Comparison of Approaches – Map File: Application (1..)						
	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	Object Name
Interface and association	162	0	16	4	0	3422 ccounter.o
	292	0	16	0	0	3696 ccounter_threadsafe.o
	148	0	48	0	0	2570 ccriticalsection.o
	110	0	24	0	0	2465 cmutex.o
	272	0	0	0	0	3602 hardwareconfiguration.o
	460	0	64	0	0	3362 led_mcbstm32f400.o
	152	0	0	0	0	2419 main_application.o
	64	26	392	0	1536	940 startup_stm32f407xx.o
	764	0	0	5	4	6913 stm32f4xx_hal.o
	1450	0	0	0	0	9221 stm32f4xx_hal_cortex.o
	1802	0	0	0	0	5263 stm32f4xx_hal_gpio.o
	3314	0	0	0	0	7606 stm32f4xx_hal_rcc.o
	398	0	24	4	0	2732 system_stm32f4xx.o

	9404	26	616	16	1544	54211 Object Totals
© MicroConsult - Microelectronics Consulting & Training GmbH						
Tuesday, October 4, 2022					S 20	MICROCONSULT

Example Resource Protection

Example Resource Protection – Comparison of Approaches – Map File: Application (...2)						
Template parameter	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	Object Name
	116	0	0	4	0	3029 ccounter.o
	46	0	0	0	0	1406 ccriticalsection.o
	46	0	0	0	0	1323 cmutex.o
	272	0	0	0	0	3601 hardwareconfiguration.o
	460	0	64	0	0	3361 led_mcbstm32f400.o
	262	0	0	0	0	5420 main_application.o
	64	26	392	0	1536	940 startup_stm32f407xx.o
	764	0	0	5	4	6912 stm32f4xx_hal.o
	1450	0	0	0	0	9220 stm32f4xx_hal_cortex.o
CRTP	1802	0	0	0	0	5262 stm32f4xx_hal_gpio.o
	3314	0	0	0	0	7605 stm32f4xx_hal_rcc.o
	398	0	24	4	0	2731 system_stm32f4xx.o
	9010	26	512	16	1544	50810 Object Totals
	Code (inc. data)	RO Data	RW Data	ZI Data	Debug	Object Name
	116	0	0	4	0	2987 ccounter.o
	46	0	0	0	0	1716 ccriticalsection.o
	46	0	0	0	0	1601 cmutex.o
	272	0	0	0	0	3587 hardwareconfiguration.o
	460	0	64	0	0	3347 led_mcbstm32f400.o
	310	0	0	0	0	6608 main_application.o
	64	26	392	0	1536	928 startup_stm32f407xx.o
	764	0	0	5	4	6898 stm32f4xx_hal.o
	1450	0	0	0	0	9206 stm32f4xx_hal_cortex.o
	1802	0	0	0	0	5248 stm32f4xx_hal_gpio.o
	3314	0	0	0	0	7591 stm32f4xx_hal_rcc.o
	398	0	24	4	0	2717 system_stm32f4xx.o
	9058	26	512	16	1544	52434 Object Totals

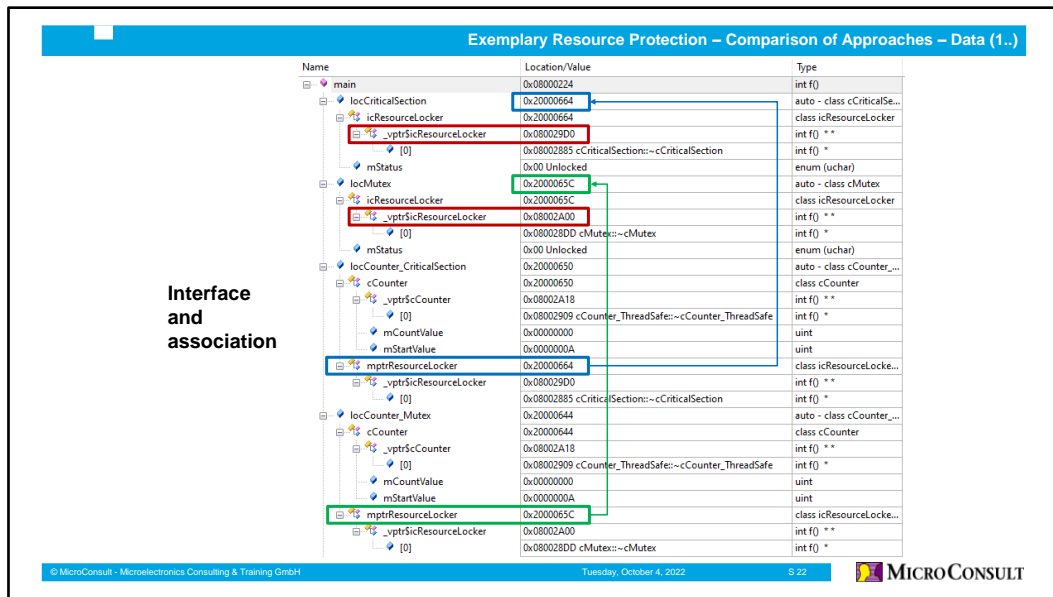
© MicroConsult - Microelectronics Consulting & Training GmbH

Tuesday, October 4, 2022

S. 21



3.6.3 Data



Example Resource Protection


Name	Location/Value	Type
main	0x08000224	int f()
locCounter_CriticalSect...	0x20000664	auto - tcCounter_ThreadSafe<ResourceLocker::cCriticalSection> < class cCriticalSection >
cCounter	0x20000664	class cCounter
mCountValue	0x0000000B	uint
mStartValue	0x0000000A	uint
mResourceLocker	0x20000668	class cCriticalSection
mStatus	0x00 Unlocked	enum (uchar)
locCounter_Mutex	0x2000065C	auto - tcCounter_ThreadSafe<ResourceLocker::cMutex> < class cMutex >
cCounter	0x2000065C	class cCounter
mCountValue	0x0000000B	uint
mStartValue	0x0000000A	uint
mResourceLocker	0x20000660	class cMutex
mStatus	0x00 Unlocked	enum (uchar)

Name	Location/Value	Type
main	0x08000224	int f()
locCounter_CriticalSection	0x20000664	auto - tcCounter_ThreadSafe<ResourceLocker::tcResourceLocker<ResourceLocker::CriticalSection> > < tcResourceLocker<ResourceLocker::CriticalSection> >
cCounter	0x20000664	class cCounter
mCountValue	0x0000000B	uint
mStartValue	0x0000000A	uint
mResourceLocker	0x20000668	tcResourceLocker<ResourceLocker::cCriticalSection> < class cCriticalSection >
locCounter_Mutex	0x2000065C	auto - tcCounter_ThreadSafe<ResourceLocker::tcResourceLocker<ResourceLocker::cMutex> > < tcResourceLocker<ResourceLocker::cMutex> >
cCounter	0x2000065C	class cCounter
mCountValue	0x0000000B	uint
mStartValue	0x0000000A	uint
mResourceLocker	0x20000660	tcResourceLocker<ResourceLocker::cMutex> < class cMutex >


© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 5:23 MICROCONSULT

3.6.4 Trace

Exemplary Resource Protection – Comparison of Approaches – Trace			
Functions	Interface and association	Template parameter	CRTP
	Amount of assembly instructions		
Constructor call for object: <code>locCounter_A</code> <code>locCounter_CriticalSection</code>	35	33	22
<code>cCounter_ThreadSafe::setCountToStartValue()</code>	53	39	53
<code>cCounter::count()</code>	9	9	9

© MicroConsult - Microelectronics Consulting & Training GmbH Tuesday, October 4, 2022 S 24  MICROCONSULT

3.6.5 Summary

Exemplary Resource Protection – Comparison of Approaches – Summary			
Aspects	Interface and association	Template parameter	C RTP
Expansion e.g. with semaphore	New class <code>cSemaphore</code> , implementing the interface and instance	New class <code>cSemaphore</code> with template class instance of <code>cCounter_</code> <code>ThreadSafe</code>	New class <code>cSemaphore</code> with template class instance of <code>cCounter_</code> <code>ThreadSafe</code>
Polymorphism (binding)	Dynamic	Static	Static
Protection mechanism exchangeable at runtime	Yes	No	No
VMT (virtual method table) (→ overhead and risk)	Yes	No	No
VMT and interface pointer (→ overhead and risk)	Yes	No	No
Program and data memory consumption	Low	Lowest	Lower
Runtime performance	Fast	Fastest	Faster
<div> <div>© MicroConsult - Microelectronics Consulting & Training GmbH</div> <div>Tuesday, October 4, 2022</div> <div>S. 29</div> <div> MICROCONSULT</div> </div>			

4 Summary and Outlook

Summary and Outlook

- Depending on your application requirements, **decide considerably** between dynamic and static polymorphism (a mix is also possible) - already at software **architecture level**.
- The more **safety** critical an application is, the stronger we recommend **static polymorphism**.
- With newer C++ constructs like `std::function` or `std::variant`, polymorphism can also be implemented. However, exception handling is required to prevent runtime errors!
- Download: <http://download.microconsult.net/ese2022/polymorphism.zip>
- My ESE presentations from the last two years cover similar topics:
Download: <http://download.microconsult.net/ese2020/interface-designs.zip>
Download: <http://download.microconsult.net/ese2021/port-designs.zip>

© MicroConsult - Microelectronics Consulting & Training GmbH
Tuesday, October 4, 2022
S 26
MICROCONSULT

5 MicroConsult Training and Coaching on Polymorphism

MicroConsult Training and Coaching on Polymorphism

Training [English and German]:

[Software Architectures for Embedded and Real-Time Systems](#)
[English](#) [German](#)

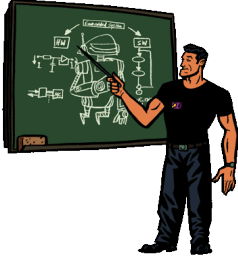
[Embedded C++: Object-Oriented Programming for \$\mu\$ C with C++/EC++](#)
[English](#) [German](#)

[Embedded C++ Advanced: Object-Oriented Programming for \$\mu\$ C with C++/EC++](#)
[English](#) [German](#)

[Embedded Software Design and Patterns with C](#)
[English](#) [German](#)

Coaching [English and German]:
[Please contact us with your topic.](#)


MicroConsult GmbH
Thomas Batt Dipl.-Ing. (FH)
Senior Manager Training & Coaching
t.batt@microconsult.com
+49 (0)89 450617-35
www.microconsult.de



© MicroConsult - Microelectronics Consulting & Training GmbH

Tuesday, October 4, 2022

§ 27

 **MICROCONSULT**